Special Purpose Computer Architectures for High Speed Optimisation

Abramson, David; de Silva, A; Randall, Marcus; Postula, Adam

Published in:
Proceedings of the Second Australasian Conference on Parallel and Real Time Systems

Special Purpose Computer Architectures for High Speed Optimisation

D. Abramson †
A. de Silva †
M. Randall †
A. Posutla §

†School of Computing and Information Technology
Griffith University
Kessels Rd, Brisbane,
Queensland, 4111
davida@cit.gu.edu.au
Phone: +61-7-875 5049
Fax: +61-7-875 5051

§ Department of Electrical Engineering
University of Queensland

Abstract

This paper discussed two computationally intensive optimisation algorithms for 0-1 integer programs, namely simulated annealing and branch and bound. It then describes an application specific computing platform designed to accelerate their performance. The paper justifies the general approach and gives details of the algorithms.

1. Introduction

Optimisation is found in many fields, and takes many different forms. Linear programming was first used by Dantzig in 1948 for solving optimisation problems involving linear cost functions and linear constraints. The traditional method for solving such problems has been the Simplex algorithm [1], however, Interior Point methods [2] are now attracting much attention for large problems. Integer programming is used extensively for scheduling activities. Exact algorithms such as branch-and-bound, and heuristic methods such as genetic algorithms and simulated annealing have been used with success. However, large optimisation problems can take a long time to solve on conventional workstations. There are many examples in the Oil and Transportation industries of problems taking hours to solve even on super computers. Optimal decisions produced in a timely manner in these industries alone amount to millions of dollars.

Accordingly, effort has been focused on methods to accelerate such techniques using general purpose vector and parallel super computers. Unfortunately, in many cases, the speedup has been disappointing. Various researchers report vector speedups of the order of only 1.5 times [3,4]. Further, even when reasonable speedup is achieved, the hardware cost may be so high that it is not possible to dedicate the computer to the one task for sufficient amounts of time.

Unlike general purpose computers, Application Specific Architectures are computers that only solve a narrow range of problems. Their specific nature means that they are often much simpler than general purpose computers, and more importantly, they can provide very fast solutions to problems which require large amounts of computation. Thus, they can offer a large improvement in the price/performance characteristics of general purpose computers. Application specific architectures are an ideal platform for acceleration of optimisation algorithms because knowledge of the algorithm and data structure can be incorporated into the machine. However, up to date, Application

Specific Architectures have been difficult to build because they require real hardware to be constructed for each new problem, and thus have not been widely used for general purpose scientific computations. Thus, a machine designed for acceleration of the simplex method is unlikely to be usable for simulated annealing.

Technology is now available in the form of Field Programmable Gate Arrays (FPGAs) which allows hardware to be reconfigured without any physical modification. This offers enormous potential because it should now be possible to build a reconfigurable special purpose architecture which can be used to solve a range of problems. Thus, a reconfigurable processor can be attached to a conventional workstation and be loaded with the correct logic for accelerating the current task.

In this paper we explore the concept of using FPGAs to build computer architectures for providing very high performance on certain real and integer optimisation algorithms. Specifically, we will look at algorithms such as General Simulated Annealing and Branch-and-bound, both is which have been used to solve large integer optimistion problems.

2. Feasibility of using application specific architectures

Low speedup has been reported on the vectorisation and parallelisation of some optimisation algorithms. One reason for this is that many problems involve sparse data, and thus the concurrency is typically quite low. For example, modern forms of the Simplex method have been optimised to remove all of the redundant work. This means that the resultant algorithm is not only sequential but has almost no concurrency at each sequential step [5]. Thus, vector machines are inefficient because the vectors are too short to overcome the internal pipeline latency, and similarly, parallel machines cannot utilise sufficient processing elements to achieve reasonable speedup. Further, general purpose machines spend a significant portion of their time manipulating the sparse data structures required to hold the information, resulting in many machine cycles being devoted to housekeeping activities rather than useful work.

There are some algorithms which are inherently concurrent, and thus can be parallelised (but not necessarily vectorised). For example, simulated annealing can often be parallelised by allowing more than one move to occur concurrently. Genetic algorithms can be parallelised by allowing multiple mating operations to happen at the same time. In the case of simulated annealing, parallelisation can involve many complex synchronisation operations. Genetic algorithms can involve large amounts of interprocessor communication due to broadcasting the population to all processors. Even when reasonable speedup is achieved, the cost of parallel hardware can prevent their practical commercial use.

The branch-and-bound algorithm appears at first inspection to be an ideal parallel application because it involves searching a tree of potential solutions. Performance of this algorithm on parallel machines is variable, depending on the nature of the pruning and workload allocation heuristics. Some anomalies arise in parallel solutions in which more nodes of the tree are searched than in their sequential counterparts. In [8] a number of studies have shown excellent speedup, however there are still many difficulties reported [9]. In [10] a new machine design is proposed to account for some of these difficulties.

In 1991 Abramson described a special purpose machine for solving an integer programming problem using simulated annealing [6]. The motivation for constructing this machine was that simulated annealing, whilst a powerful general purpose optimisation algorithm, is extremely slow. In this case, realistic problems could take weeks to solve on conventional workstations. Further, the Monte Carlo nature of the algorithm meant that it was not possible to vectorise large portions of the code, and the performance on super computers was not much better than workstations. A parallel

version of the program achieved moderate speed ups on a shared memory machine, but the price/performance of the solution was too large for commercial exploitation. The hardware which was built executed the code about 100 times faster than the same program running on a CRAY Y/MP. Interestingly, this machine gained its performance from two main sources. First, it utilised very low level concurrency which cannot be extracted by vector and parallel computers. Second, it avoided all address arithmetic normally required for matrix manipulation. Complete details of the implementation are available in [6] and [11]. The board was implemented using conventional logic devices and was hosted by a PC or workstation and cost a few thousand dollars. It was controlled by a simple finite state machine which implemented the annealing algorithm, and contained no fast logic or pipelined stages. The latter techniques would have enhanced the performance by another order of magnitude.

Recent work by Abramson in the area of numerical optimisation, for both real valued and integer valued problems, has discovered that it is extremely difficult to gain much speedup in the simplex method using vector and parallel machines. As mentioned previously, this is primarily due to the sparse nature of the data structures and the very high efficiency of the modern simplex codes. Even the newer interior point algorithms do not seem to achieve much speedup [7].

The specific architecture described in [6] is not general enough to handle more than one problem class. It was constructed from conventional logic devices, and thus cannot be modified easily. The work described in this paper will allow a machine to be constructed which can be targeted to a number of different optimisation algorithms.

3. General Simulated Annealing

Consider the 0-1 integer programming problem

$$\min \quad C(X) = \sum_{j \in J} C_j x_j$$

$$\text{Subject to} \quad AX \otimes b,$$

$$x_j \in \{0,1\}.$$

Where, $\otimes$ is either $\leq$, $=$, $\geq$ or a mixture of these,

$X = x_j, j \in J$,

A is an $(m \times n)$ matrix,

b is an m-vector.

The general purpose SA algorithm for solving the above problem starts with an initial configuration, its associated cost, and a given starting temperature. At each subsequent temperature, the algorithm iterates by perturbing the current configuration an appropriate number of times, called the Markov chain length. Perturbation is simulated by toggling the value of a randomly chosen binary variable x_j from 0 to 1 or vice versa. It is possible that, by changing the value of chosen variable x_j we may obtain an infeasible solution. We may, at this stage, accept the infeasible solution with a resulting penalty, or restore feasibility. The new configuration is accepted if the change in cost is negative. Otherwise, the acceptance probability is used to determine whether the uphill move is accepted. The mechanism for transforming the current solution into one of its neighbourhood is called a *Markovian transformation*, in which, the current configuration only depends on the immediately previous one. The temperature is then decreased according to one of a number of cooling schedules [13]. The process is continued until there is no change in cost between a number of consecutive Markov chains.

The basic form of the SA algorithm may be represented as follows

Generate initial configuration call X
Compute cost of initial configuration $C(X)$
Compute initial temperature T_0
Set $T = T_0$
  While not termination do begin
    Repeat Markov-Chain-Length times begin
      i = random (0, n - 1)
      Set X' = flip(X ,i)
      IF using force feasible approach, THEN
        restore feasibility of X'
      $\Delta C = C(X') - C(X)$
      IF ($\Delta C \leq 0$) or ($e^{-\Delta C/T}$ > unif_rand(0,1)), THEN
        Set $X = X'$
    end
  T = Cool (T)
  end

where flip(X , i) will flip the value of the *i'th* element of X and return the resulting vector.
random(a,b) returns a random uniformly distributed integer between a and b
unif_rand(a,b) returns a random uniformly distributed number between a and b
Cool (T) returns a value for the temperature after applying a cooling operation to T.
In the experiments reported in this paper we use a reheating scheme in which the temperature is reduced and then raised to escape from local minima

4. Branch and Bound

A popular method of solving the type of integer programs cited in the previous section is the *branch and bound* algorithm. The branch and bound algorithm can be outlined simply by the following manner. First, the integer requirements are relaxed and the resulting LP is solved. If the solution has all integer components, then this solution is optimal. If the solution does not have all integer components, a noninteger variable x_j is chosen for *branching*. Branching from x_j gives rise to two LP subproblems. The left hand child is created by adding a constraint $x_j \leq I_j$ to the parent, where I_j is the integer part of x_j. For the right hand side child, the constraint $x_j \geq I_j + 1$ is added to the parent. This will partition the solution space, such that the infeasible noninteger value is removed from the solution space, while retaining all the feasible integer solutions. The branching process is carried out recursively; each of the two subproblems will give rise to two more subproblems, which will create an enumeration tree of continuous linear programs, by partitioning the solution space. The subproblems encountered in the enumeration tree are commonly referred to as nodes. The optimal solution of the integer program will lie in some branch of the tree. Repeatedly branching as explained previously will eventually lead to a solution, which has all integer components. The value of the objective function of this subproblem, is stored as the best integer solution found so far. This value is used to prune the enumeration tree. If the objective value obtained by solving any one of the subproblems of the tree is greater than the best integer solution found so far, then the objective of all the descendants will also be greater than the best integer solution found so far. This is because extra constraints are added as we move down the tree and the optimal objective of the subproblems will only increase. Thus, if the solution of a subproblem yields a objective value greater than the best integer solution found so far, then this subproblem is pruned. Pruning of the tree in this fashion is called *bounding*. This procedure is repeated until the whole tree has been searched. Selecting the noninteger variables x_j to branch from the set of noninteger

variables and selecting a subproblem to search after the current subproblem is pruned, are two important aspects, which will have a significant impact on the performance of the branch and bound algorithm. A number of heuristics have been developed to carry out these two processes.

Usually a large number of linear programs have to be solved to solve an integer linear program and any improvement in the performance of solving LPs will have a beneficial impact in solving integer programs. The major portion of time taken in carrying out a branch and bound algorithm is taken for the solution of the linear programming subproblems. Thus, our focus in this part of the project will be acceleration techniques for solving the linear programs. One of the more common algorithms for solving these linear programs is the simplex algorithm [1]. The simplex method is almost universally favoured by most researchers, due to its superior warm starting capability. Warm starting makes it possible to re-optimise a sightly different linear program using the solution of the original linear program. Each child of the branch and bound algorithm is only slightly different from its parent and due to its superior warm starting performance, the simplex method is the preferred algorithm for solving the continuous linear programs. The search strategy described here is known as the depth-wise search. It is possible to carry out a breadth-wise search, but the depth-wise search is favoured by most researchers due to the superior warm starting capability of the simplex method. In a depth-wise search the subproblems change only slightly as the branch and bound algorithm moves down the tree.

5. Reconfigurable Logic as an implementation platform

Both of the algorithms discussed in sections 4 and 5 share the property that they can take large amounts of time to execute. However, they differ in many other important respects. The SA algorithm is only a heuristic and provides no guarantee of optimality, whereas the branch-and-bound procedure searches until an optimal solution is found. The SA algorithm works on integer only variables, whereas the branch-and-bound utilises a linear relaxation of the original problem and then uses the simplex algorithm to solve the subproblems. The latter algorithm requires floating point operations on sparse matrixes. Thus, whilst the two algorithms can potentially benefit from hardware acceleration techniques, they clearly have different architectural demands. Consequently, no single architecture will be optimal for both applications.

The hardware platform under consideration in this project provides a totally reconfigurable resource. The system, shown in Figure 1, incorporates up to 16 Xilinx Field Programmable Gate Arrays (FPGAs), each with a maximum of 10,000 gates. Further, FPGAs can be interconnected using Aptix Field Programmable Interconnection Chips (FPICS), and thus the logic of the entire system can be tailored for an individual problem. Extra breadboarding pins are available on the board to allow extra logic devices to be connected.

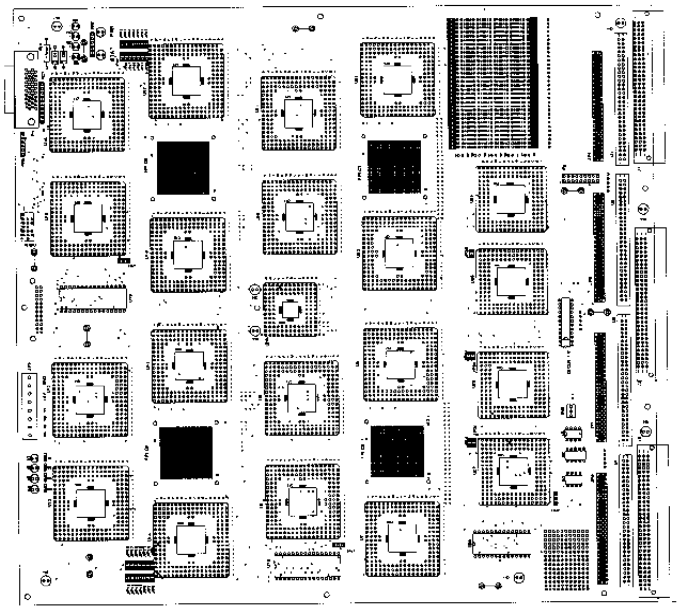

Figure 1 - Aptix Field Programmable Architecture Board

Both of the algorithms under consideration are memory intensive, and moreover, speedup through a dedicated architecture is often achieved by an appropriate memory organisation which delivers the data efficiently. However, the FPGA devices do not contain much random access memory. Accordingly, we have design and built a reconfigurable memory unit which connects to a FPGA socket. Thus, we have the choice of plugging memory or logic into any of the FPGA sockets on the board. We expect to adopt a similar approach for providing floating point support to the board.

The Aptix board is connected to a host system as shown in Figure 2. The host is responsible for down loading the connection information, as well as controlling the execution of the target algorithm. A host-interface-module is provided for connecting the serial loading chain of the FPICs to the serial port of the host. Likewise, the loading chain of the FPGAs is connected to another serial interface. It is our intension to connect the architecture to the host via a high speed parallel interface, because it will be necessary to control the loading and extraction of data at a much faster rate than required for loading the FPGAs and FPICs.

At this stage, we have only begun the design of the architectures necessary to support the two algorithms discussed in sections 3 and 4. It is clear from this initial design phase that significant speedup is only achievable by carefully mapping the data structures onto the target hardware. We are experimenting with writing the architectural specifications in VHDL and using logic synthesis to generate detailed hardware interconnections. This approach has the advantage that we can simulate the hardware description before downloading it, to verify correct execution. We can also easily compare the time to execute the algorithm on a conventional computer. However, it is not clear whether current commercial logic synthesis systems are sufficiently mature to provide efficient mappings. One of the project outcomes will be some guidelines for use of VHDL in this mode. We expect to present the results of this work at a later date.

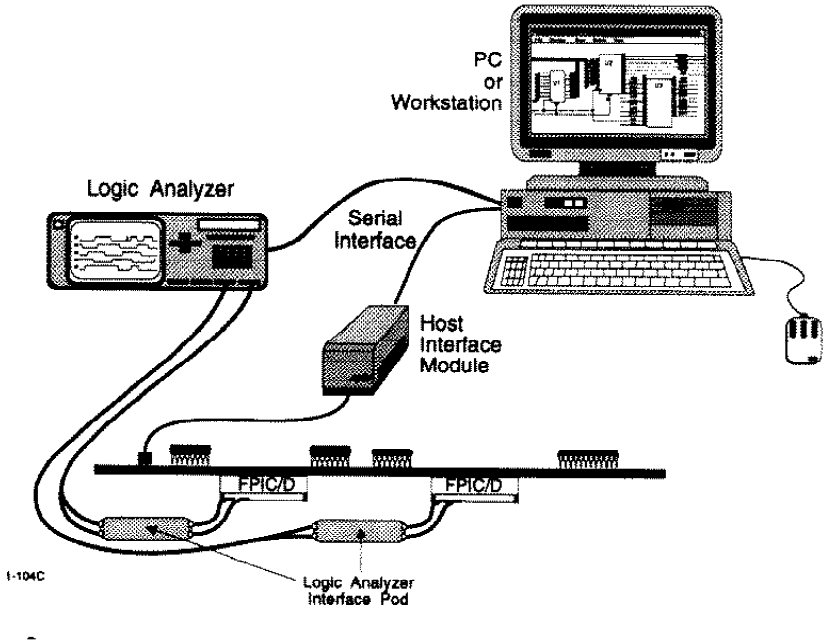

Figure 2 - driving the hardware system

6. Conclusions

This project is still in its earliest stages. We have studied two important optimisation algorithms and are currently experimenting with their effectiveness at solving large problems. At the same time, we are profiling their performance and considering the appropriate techniques for improving performance.

In the case of the simulated annealing algorithm, the current focus is in the area of feasibility restoration. Prior work in the design of a simulated annealing machine [6] has suggested that the remaining parts of the algorithm can be accelerated through implementation in hardware. However, the current feasibility restoration algorithm uses division operations on floating point numbers. We are currently exploring ways of simplifying this algorithm to use other arithmetic operators on fixed point numbers.

Likewise, most implementations of the simplex algorithm rely on floating point matrix arithmetic. In order to get substantial speedup we plan to use fixed point arithmetic where possible, and also to exploit the spare nature of the constraint matrix. If fixed point arithmetic can be used then it will be possible to implement the ALUs on the FPGAs, otherwise, external floating point hardware will be required. Managing sparse data structures will require specially designed memory units which are capable of delivering the data at a high rate.

We hope to report the results of the studies outlined in this paper at a later date.

Acknowledgments

The work described in the paper is supported by the Australian Research Council under the Large Grants Scheme. Project members include the authors plus Paul Logithetis, Zippin Fang and Ricky Handojo.

References

1 Danztig, G. B. "Linear Programming and Extentions", Princeton, Princeton University Press (1963).

2 Karmarkar, N. A "New Polynomial-Time Algorithm for Linear Programming", Combinatorica 4, pp 373-395 (1984).

3 Arthur, J. L., Frendewey J. O. and Schumichrast, R. T. "Experience with Vectorizing a Linear Programming Code" (manuscript(, Oregon State University, Corvallis, OR, 1987.

4 Beasley, J. E. "Linear Programming on CRAY Supercomputers" (manuscript), Derpartment of Management Science, Imperial College London, England, Septemper 1987.

5 James M. Ortega, Introduction to Parallel and Vector Solutions of Linear Systems, Frontiers of Computer Science Series, Plenum Press, 1988.

6 Abramson, D.A., "A Very High Speed Architecture to Support Simulated Annealing", IEEE Computer, May 1992.

7 Edwards J. Tomlin, J. "Parallel Cholesky Factorization", Proceedings of 5th Australian Supercompting Conference, pp 105 - 114, December 1992.

8 Arvindam S, Kumar V and Rao, N, "Efficient Parallel Algorithms for Search Problems: Applications in VLSI CAD", The Third Symposium on the Frontiers of Massively Parallel Computation, October 1990, University of Maryland, pp 166-169.

9 Altman E, Marsland T and Breitkreutz, "Accounting for Parallel Tree Search Overheads", International Conference on Parallel Processing, 1988, Vol 3, pp 198-201

10 Cheng K and Wang Q, "An Asynchronous Multiprocessor Design for Branch-and-Bound Algorithms", The Third Symposium on the Frontiers of Massively Parallel Computation, October 1990, University of Maryland, pp 65-68.

11 Abramson, D. A. and Dang, H. "School Timetables: A Case Study in Simulated Annealing", Applied Simulated Annealing, Lecture Notes in Economics and Mathematics Systems, Springer-Verlag, in press

12 Bertin, P, Roncin and Vuillemin, "Programmable Active Memories: A Performance Assessment", Technical Report, DEC Paris Laboratories, 1990.

13. N.E. Collins, R.W. Eglese and B.L. Golden, Simulated annealing: An annotated bibliography, American Journal of Mathematical and Management Sciences**,** 8, Nos 3 & 4, (1988)209-307.